Image Processing and Analysis

Generative models

Diffusion model

olivier.bernard@insa-lyon.fr

What is the purpose of diffusion models?

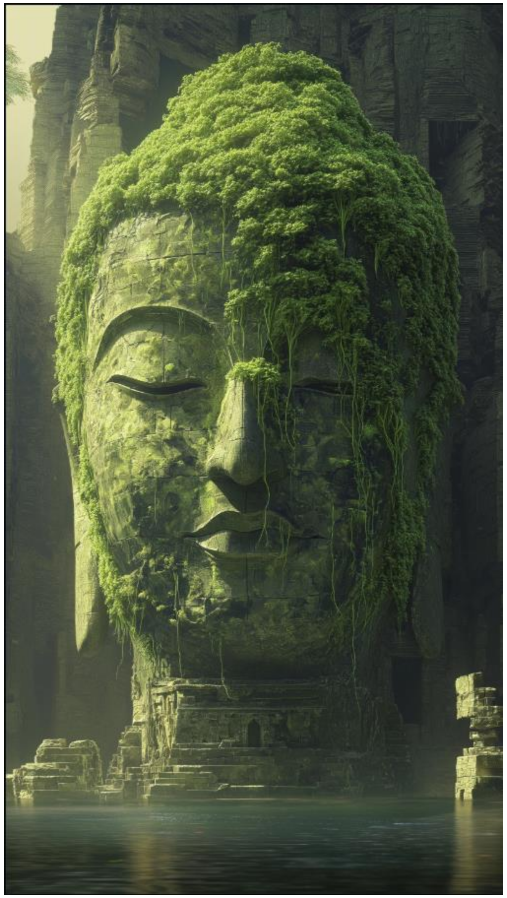
- Best current methods for synthetic image generation
- Allows generating images in a conditioned form
- Many software solutions, such as Midjourney, DALL-E

An Asian girl in ancient coarse linen clothes rides a giant panda and carries a wooden cage. A chubby little girl with two buns walks on the snow. High-precision clothing texture, real tactile skin, foggy white tone, low saturation, retro film texture, tranquil atmosphere, minimalism, long-range view, telephoto lens

What is the purpose of diffusion models?

- Best current methods for synthetic image generation
- Allows generating images in a conditioned form
- Many software solutions, such as Midjourney, DALL-E

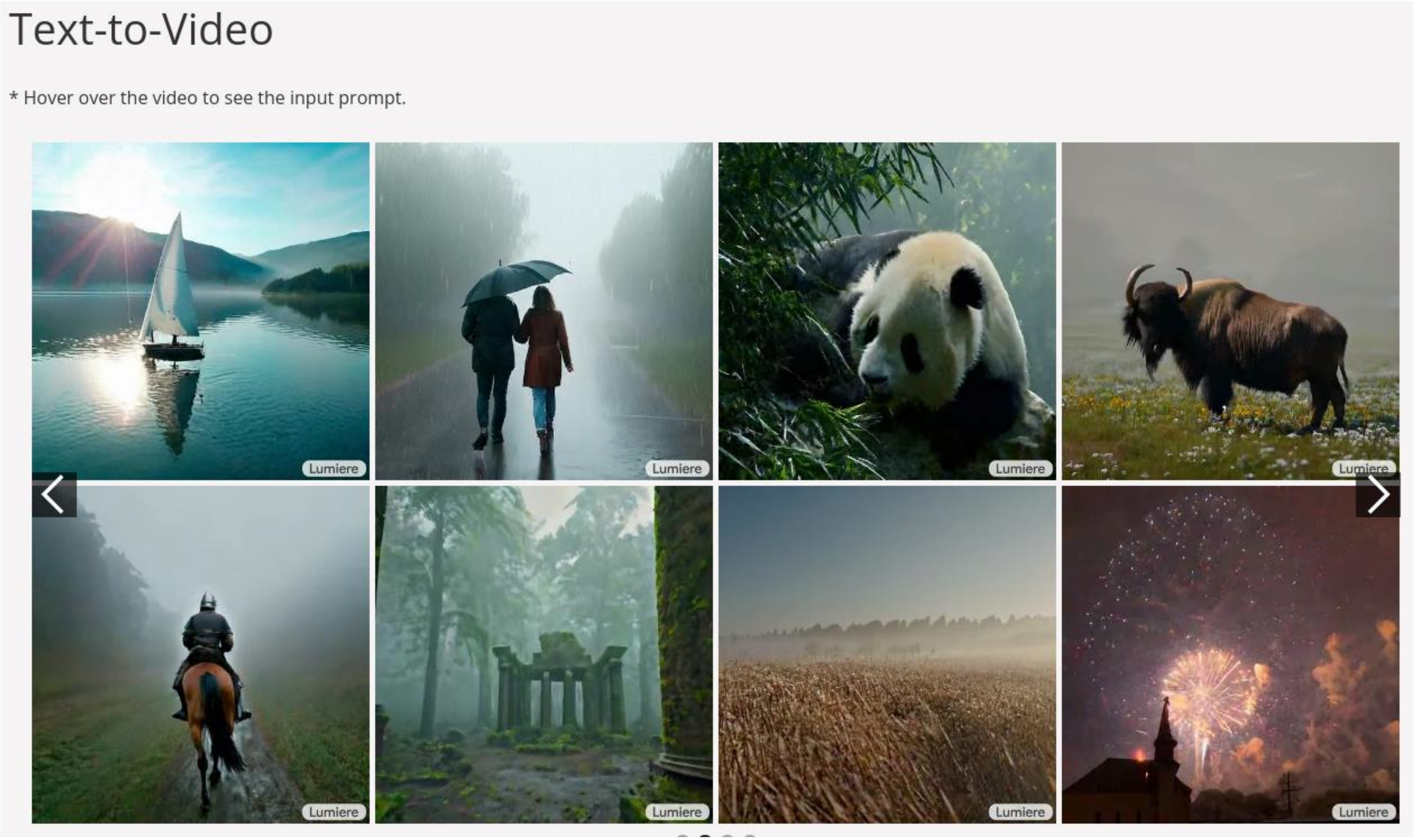
A digital artwork depicting the Buddha's head, intricately designed with green trees growing from it and vines surrounding its face. The background is an enchanted forest filled with ancient ruins, creating a mystical atmosphere. In front of the Buddha's head lies a tranquil river that reflects his serene expression. This scene embodies peace amidst chaos in nature



What is the purpose of diffusion models?

- Recent extensions for video synthesis

  https://lumiere-video.github.io/#section_image_to_video



What is the purpose of diffusion models?

- Recent extensions for video synthesis

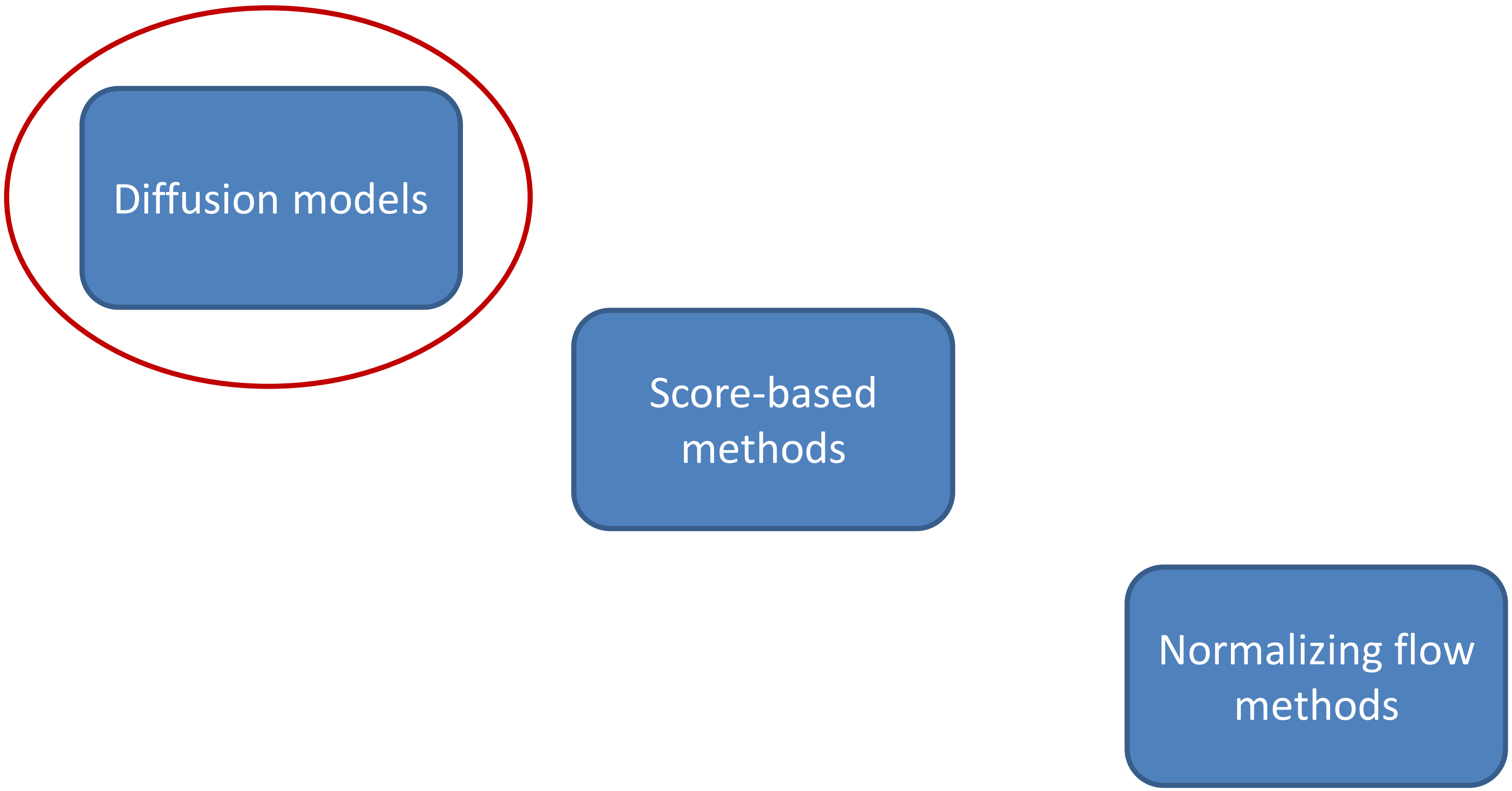
  https://lumiere-video.github.io/#section_image_to_video

Image-to-Video

* Hover over the video to see the input image and prompt.

What is the purpose of diffusion models?

- Family of diffusion networks



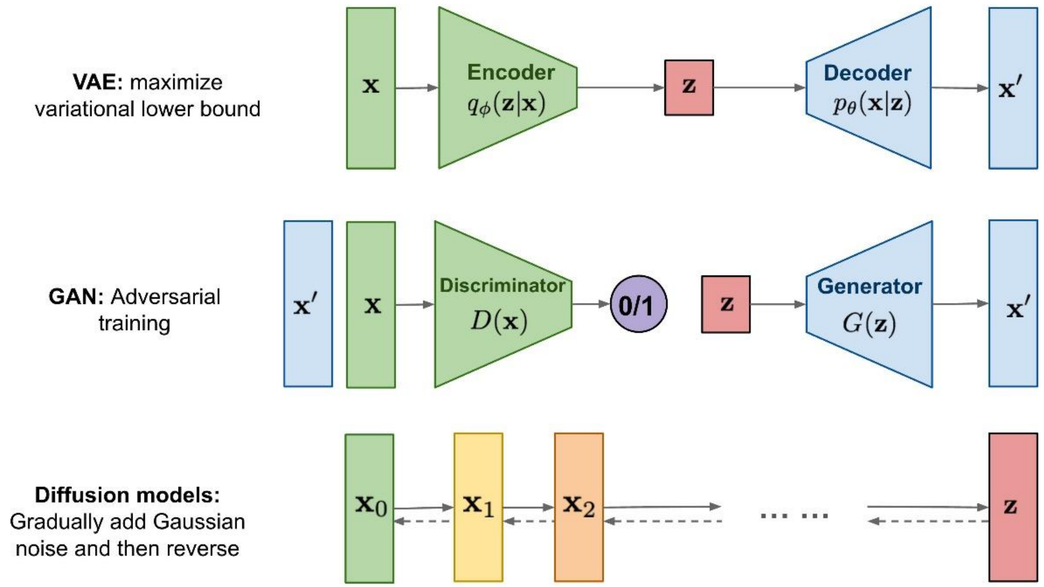
The denoising diffusion probabilistic models

DDPM

All the mathematics are described in the following blog

https://creatis-myriad.github.io/tutorials/2023-11-30-tutorial-ddpm.html

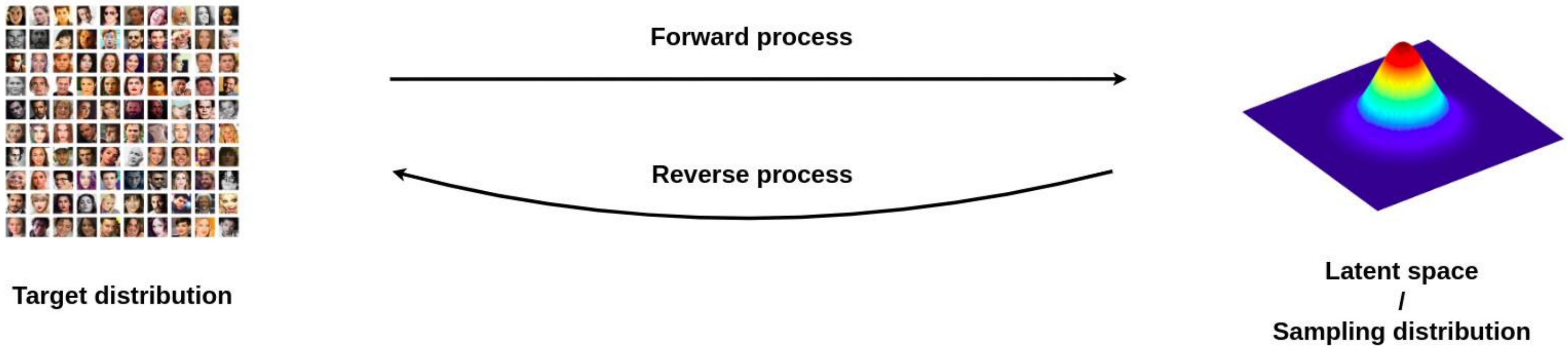
- Key characteristics
  - ➔ Belongs to the family of generative models (like VAEs)

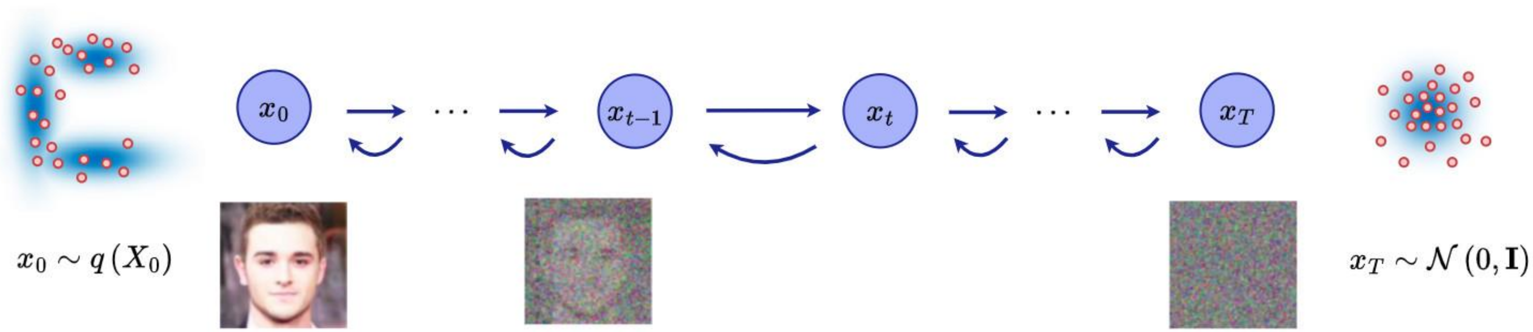


  - ➔ Based on the concept of a Markov chain

Mathematical model used to describe a system that evolves randomly between different states, following certain probability rules

- Key characteristics
  - Define a Markov chain of diffusion steps to slowly add random noise to the data
  - The model then learns to reverse the diffusion process to construct data samples from the noise





Mathematical tools used

- Bayes' theorem

$$q(x_t \mid x_{t-1}) = \frac{q(x_{t-1} \mid x_t)\, q(x_t)}{q(x_{t-1})}$$

$$q(x_{t-1} \mid x_t) = \frac{q(x_t \mid x_{t-1})\, q(x_{t-1})}{q(x_t)}$$

- Marginal theorem

$$q(x_0, x_1, \cdots, x_T) = q(x_{0:T})$$

$$q(x_0) = \int q(x_0, x_1, \cdots, x_T)\, dx_1 \cdots dx_T$$

$$q(x_0) = \int q(x_{0:T})\, dx_{1:T}$$

- Theorem of conditional probabilities

$$q(x_{t-1}, x_t) = q(x_t \mid x_{t-1}) q(x_{t-1})$$

$$q(x_{1:T} \mid x_0) = q(x_T \mid x_{0:T-1}) q(x_{T-1} \mid x_{0:T-2}) \ldots q(x_1 \mid x_0)$$

The probability of each event depends only on the state reached during the previous event

- Theorem of conditional probabilities

$$q(x_T \mid x_{0:T-1}) = q(x_T \mid x_{T-1})$$

$$q(x_{1:T} \mid x_0) = \prod_{t=1}^{T} q(x_t \mid x_{t-1})$$

- Bayes' theorem

$$q(x_t \mid x_{t-1}) = q(x_t \mid x_{t-1}, x_0) = \frac{q(x_{t-1} \mid x_t, x_0)\, q(x_t \mid x_0)}{q(x_{t-1} \mid x_0)}$$

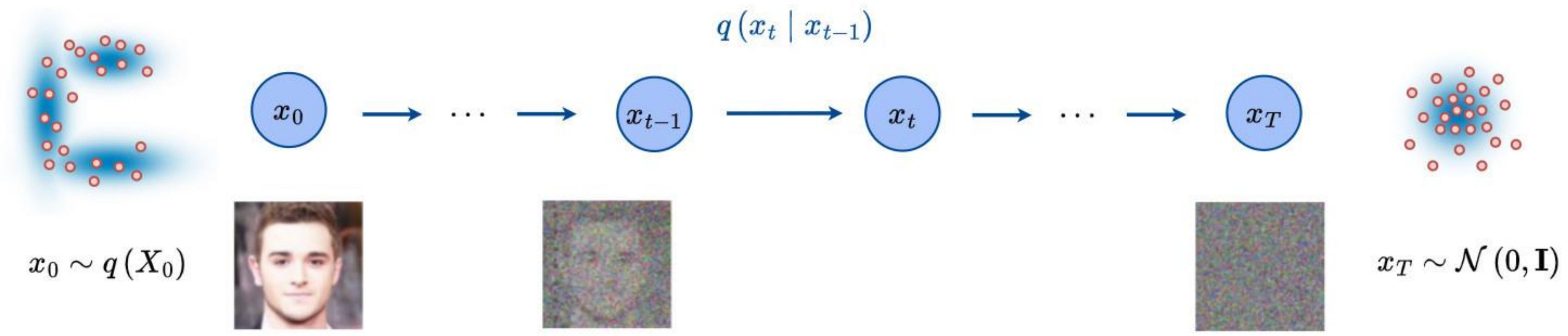
- Joint distribution

$$p_\theta(x_{0:T}) = p_\theta(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1} \mid x_t)$$

DDPM

Forward diffusion process

A procedure in which a small amount of Gaussian noise is added to the initial sample x_0, producing a sequence of noisy samples $x_1, \cdots, x_T$



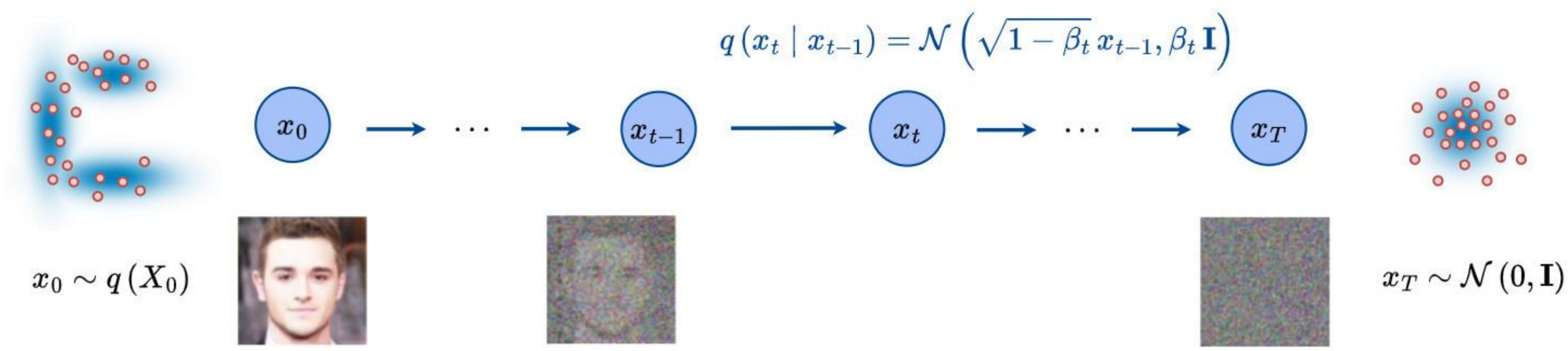
- x_0 is a sample drawn from a real data distribution $x_0 \sim q(X_0)$
- $q(x_t|x_{t-1})$ models the probability of having the state x_t given the state x_{t-1}

Forward diffusion process

The forward process of a DDPM is a Markov chain

- The prediction at step t depends only on the state at step $t-1$, which gradually adds Gaussian noise to the data x_0
- The complete process is modeled by : $q(x_{1:T} \mid x_0) = \prod_{t=1}^{T} q(x_t \mid x_{t-1})$
- The conditional probability can be effectively modeled by

$$q\left(x_t \mid x_{t-1}\right) = \mathcal{N}\left(\left(\sqrt{1-\beta_t}\right) x_{t-1}, \beta_t \mathbf{I}\right)$$



Forward diffusion process

- How to define the variance β_t ?

  - ➔ $\{\beta_t \in (0,1)\}_{t=1}^{T}$ sequence of linearly increasing constants
  - ➔ $\beta_t = clip\left(1 - \frac{\bar{\alpha}_t}{\bar{\alpha}_{t-1}}, 0.999\right)$ sequence of cosine-type constants

    with $\bar{\alpha}_t = \frac{f(t)}{f(0)}$ and $f(t) = cos\left(\frac{\frac{t}{T}+s}{1+s} \cdot \frac{\pi}{2}\right)^2$

  - ➔ In this case

$$q\left(x_t \mid x_{t-1}\right) = \mathcal{N}\left(\left(\sqrt{1-\beta_t}\right) x_{t-1}, \beta_t \mathbf{I}\right)$$

$$\begin{array}{llll} \text{if} & \beta_t = 0, & \text{then} & q(x_t \mid x_{t-1}) = x_{t-1} \\ \text{if} & \beta_t = 1, & \text{then} & q(x_t \mid x_{t-1}) = \mathcal{N}(0, \mathbf{I}) \end{array}$$

Forward diffusion process

- Conditional probability: important relation

→ Using the reparameterization trick

$$q(x_t \mid x_{t-1}) = \mathcal{N}\left(\sqrt{1-\beta_t}\, x_{t-1}, \beta_t \mathbf{I}\right)$$

$$x_t = \sqrt{1-\beta_t}\, x_{t-1} + \sqrt{\beta_t}\, \epsilon_{t-1}$$

with $\epsilon_{t-1} = \mathcal{N}(0, \mathbf{I})$

→ One can demonstrate the following relation

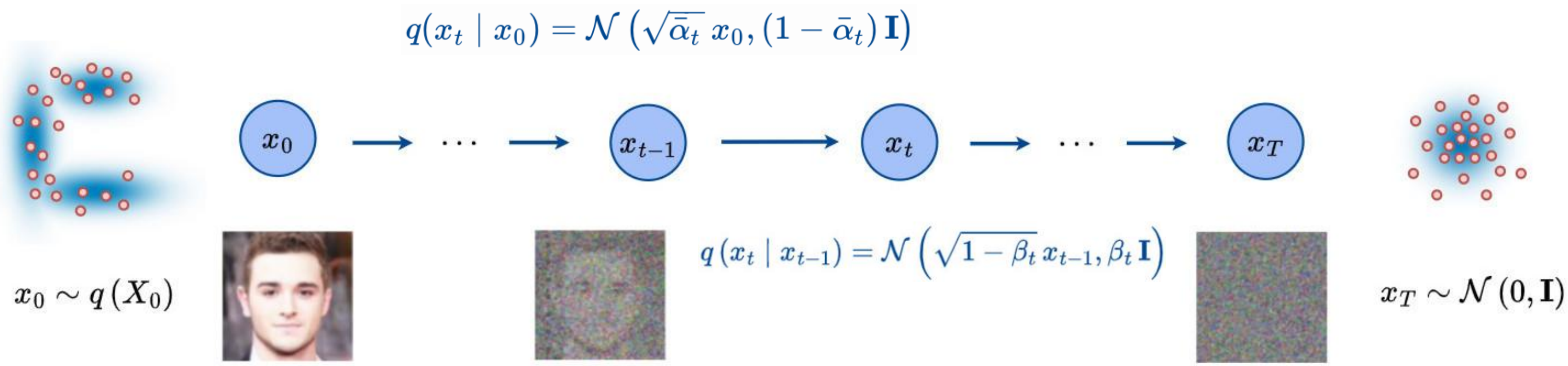
$$x_t = \sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1-\bar{\alpha}_t}\, \epsilon_t$$

$$q(x_t \mid x_0) = \mathcal{N}\left(\sqrt{\bar{\alpha}_t}\, x_0, (1-\bar{\alpha}_t)\, \mathbf{I}\right)$$

with $\alpha_t = 1 - \beta_t$

$$\bar{\alpha}_t = \prod_{k=1}^{t} \alpha_k$$

- To summarize



$$q(x_t \mid x_{t-1}) = \mathcal{N}\left((\sqrt{1-\beta_t})\, x_{t-1}, \beta_t\,\mathbf{I}\right)$$

$$\begin{aligned} &\text{if} \quad \beta_t = 0, \quad \text{then} \quad q(x_t \mid x_{t-1}) = x_{t-1} \\ &\text{if} \quad \beta_t = 1, \quad \text{then} \quad q(x_t \mid x_{t-1}) = \mathcal{N}(0, \mathbf{I}) \end{aligned}$$

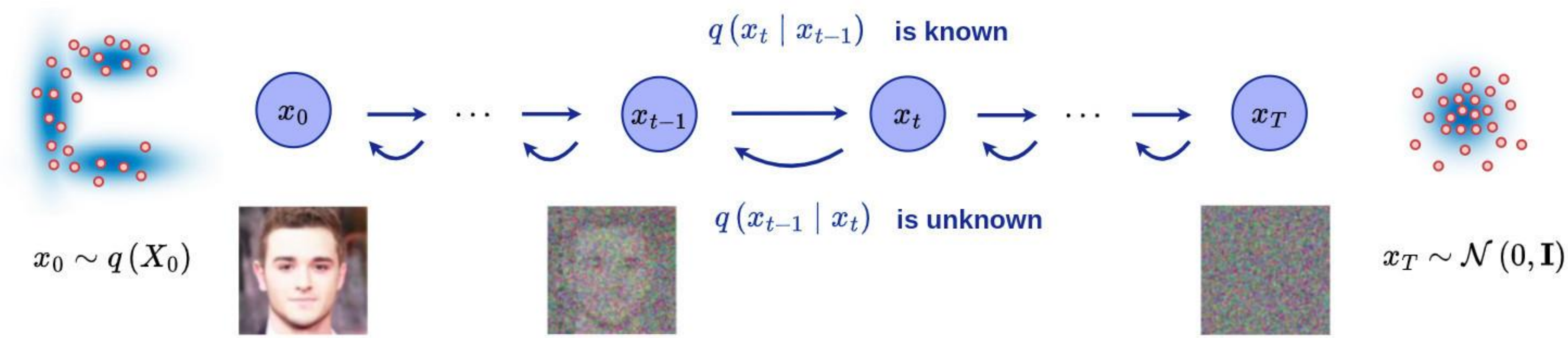
$$q(x_t \mid x_0) = \mathcal{N}\left(\sqrt{\bar{\alpha}_t}\, x_0, (1-\bar{\alpha}_t)\,\mathbf{I}\right)$$

with $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{k=1}^{t} \alpha_k$

DDPM

Reverse process

If we are able to reverse the diffusion process from $q(x_{t-1}|x_t)$, then we can generate a sample x_0 from Gaussian noise $x_T \sim \mathrm{N}(0, \boldsymbol{I})$



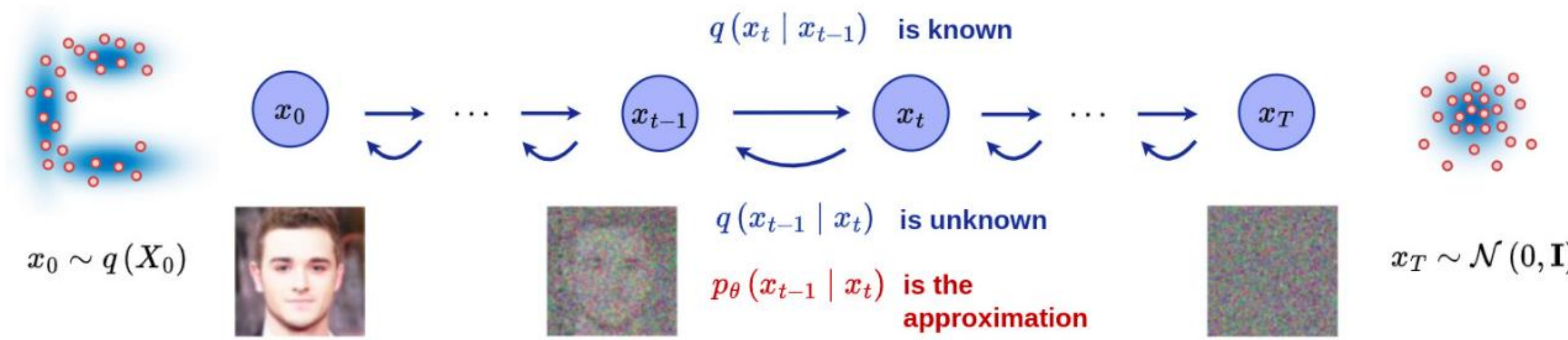
- Thanks to Bayes' theorem

$$q(x_{t-1} \mid x_t) = \frac{q(x_t \mid x_{t-1})\, q(x_{t-1})}{q(x_t)}$$

- Since $q(x_t)$ is unknown, $q(x_{t-1}|x_t)$ is intractable

Reverse process

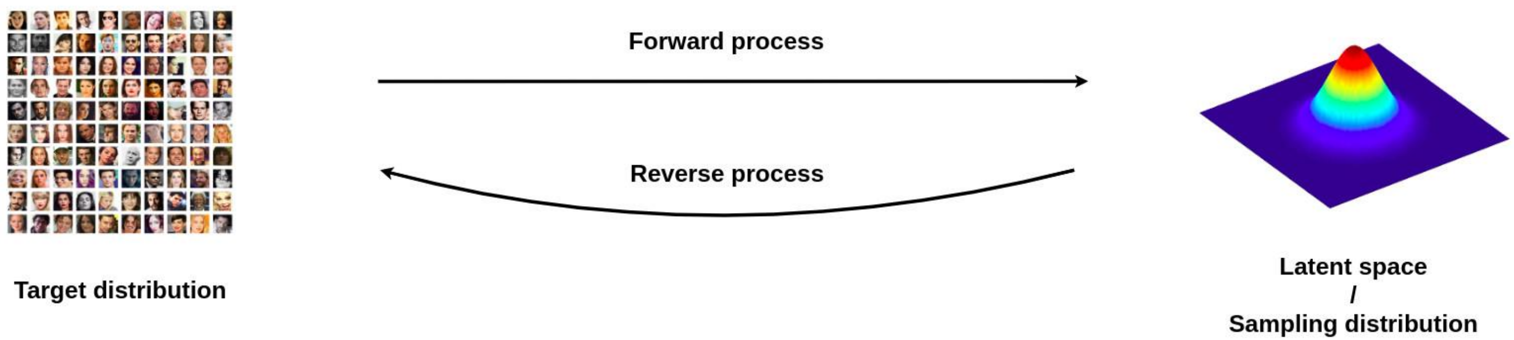
We will train a model $p_\theta(x_{t-1}|x_t)$ to approximate $q(x_{t-1}|x_t)$ in order to execute the reverse diffusion process

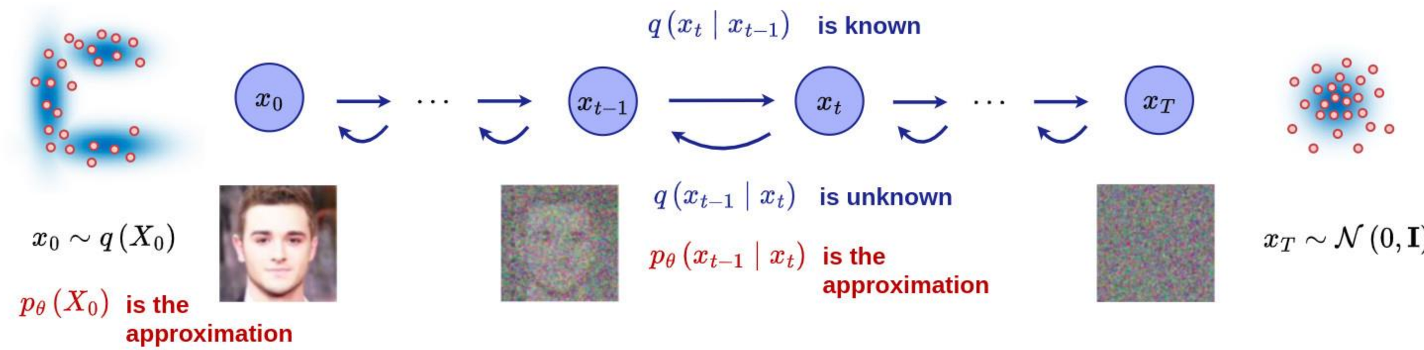


- Gaussian assumption $$p_\theta(x_{t-1} \mid x_t) = \mathcal{N}(\mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$$
- Modeling the entire reverse process

$$p_\theta(x_{0:T}) = p_\theta(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1} \mid x_t)$$

- To summarize





→ Model to learn

$$p_\theta(x_{t-1} \mid x_t) = \mathcal{N}(\mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$$

→ Entire reverse process

$$p_\theta(x_{0:T}) = p_\theta(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1} \mid x_t)$$

DDPM

Learning strategy

- Minimizing the cross-entropy between $q(x_0)$ and $p_\theta(x_0)$ results in the two distributions being as close as possible

$$H(q, p_\theta) = -\int q(x_0) \cdot \log(p_\theta(x_0))\, dx_0 = -\mathbb{E}_{x_0 \sim q}\left[\log(p_\theta(x_0))\right]$$

- Rewriting this expression using the marginal theorem

$$\begin{aligned} H(q, p_\theta) &= -\mathbb{E}_{x_0 \sim q}\left[\log\left(\int p_\theta(x_{0:T})\, d_{x_{1:T}}\right)\right] \\ &= -\mathbb{E}_{x_0 \sim q}\left[\log\left(\int q(x_{1:T} \mid x_0) \frac{p_\theta(x_{0:T})}{q(x_{1:T} \mid x_0)}\, d_{x_{1:T}}\right)\right] \\ &= -\mathbb{E}_{x_0 \sim q}\left[\log\left(\mathbb{E}_{x_{1:T} \sim q(x_{1:T} \mid x_0)}\left[\frac{p_\theta(x_{0:T})}{q(x_{1:T} \mid x_0)}\right)\right]\right] \end{aligned}$$

Learning strategy

- Jensen's inequality

$$\phi(\mathbb{E}[X]) \leq \mathbb{E}[\phi(X)]$$

$$\begin{aligned}
\rightarrow \quad H(q, p_\theta) &\leq -\mathbb{E}_{x_0 \sim q}\,\mathbb{E}_{x_{1:T} \sim q(x_{1:T}|x_0)}\left[\log\left(\frac{p_\theta(x_{0:T})}{q(x_{1:T} \mid x_0)}\right)\right] \\
&\leq -\mathbb{E}_{x_{0:T} \sim q(x_{0:T})}\left[\log\left(\frac{p_\theta(x_{0:T})}{q(x_{1:T} \mid x_0)}\right)\right] \\
&\leq \mathbb{E}_{x_{0:T} \sim q(x_{0:T})}\left[\log\left(\frac{q(x_{1:T} \mid x_0)}{p_\theta(x_{0:T})}\right)\right] \\
&\leq \mathcal{L}_{VUB}
\end{aligned}$$

- Variational upper bound

$$\mathcal{L}_{VUB} = \mathbb{E}_{x_{0:T} \sim q(x_{0:T})}\left[\log\left(\frac{q(x_{1:T} \mid x_0)}{p_\theta(x_{0:T})}\right)\right]$$

Since $H(q, p_\theta)$ is positive, minimizing $\mathcal{L}_{VUB}$ is equivalent to minimizing $H(q, p_\theta)$

- Minimizing $\mathcal{L}_{VUB}$

$$\begin{aligned}\mathcal{L}_{VUB} &= \mathbb{E}_{x_{0:T}\sim q}\left[\log\left(\frac{q(x_T \mid x_0)}{p_\theta(x_T)}\right) + \sum_{t=2}^{T}\log\left(\frac{q(x_{t-1} \mid x_t, x_0)}{p_\theta(x_{t-1} \mid x_t)}\right) - \log(p_\theta(x_0 \mid x_1))\right] \\ &= \underbrace{D_{KL}\left(q(x_T \mid x_0) \parallel p_\theta(x_T)\right)}_{\mathcal{L}_T} + \sum_{t=2}^{T}\underbrace{D_{KL}\left(q(x_{t-1} \mid x_t, x_0) \parallel p_\theta(x_{t-1} \mid x_t)\right)}_{\mathcal{L}_{t-1}} - \underbrace{\log(p_\theta(x_0 \mid x_1))}_{\mathcal{L}_0}\end{aligned}$$

The derivation of this expression is described in the following blog

https://creatis-myriad.github.io/tutorials/2023-11-30-tutorial-ddpm.html

- Minimizing $\mathcal{L}_{VUB}$

  - Remark n°1: Since the sequence $\{\beta_t\}_{t\in[1,T]}$ is chosen in advance, $q(x_T|x_0)$ is deterministic, and $\mathcal{L}_T$ is a constant term that will be ignored in the minimization process

  - Remark n°2: L_0 can be modeled by a specific decoder, or omitted for the sake of simplicity

  - Remark n°3: Using the reparameterization trick, $q(x_{t-1}|x_t, x_0)$ can be reformulated as

$$q(x_{t-1} \mid x_t, x_0) = \mathcal{N}(\tilde{\mu}_t(x_t, x_0), \tilde{\beta}_t \cdot \mathbf{I})$$

with

$$\tilde{\mu}_t(x_t, x_0) = \frac{1}{\sqrt{\alpha_t}}(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_t)$$

$$\tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t} \cdot \beta_t \qquad \bar{\alpha}_t = \prod_{k=1}^{t} \alpha_k \qquad \alpha_t = 1 - \beta_t$$

- Minimizing $\mathcal{L}_{VUB}$

  - ➔ Minimizing $\mathcal{L}_{VUB}$ thus corresponds to minimizing $D_{KL}(q(x_{t-1}|x_t, x_0) \,||\, p_\theta(x_{t-1}|x_t))$ for all time steps t

with
$$\begin{cases} q(x_{t-1} \mid x_t, x_0) = \mathcal{N}(\tilde{\mu}_t(x_t, x_0), \tilde{\beta}_t \cdot \mathbf{I}) \\ p_\theta(x_{t-1} \mid x_t) = \mathcal{N}(\mu_\theta(x_t, t), \Sigma_\theta(x_t, t)) \end{cases}$$

  - ➔ We want to make the two Gaussian distributions $q(x_{t-1}|x_t, x_0)$ and $p_\theta(x_{t-1}|x_t)$ as close as possible

  - ➔ For the sake of simplicity, we choose $\Sigma_\theta(x_t, t) = \sigma_t \mathbf{I} = \tilde{\beta}_t \mathbf{I}$

The idea is to focus on the means of the two distributions and train a neural network μ_θ to predict $\tilde{\mu}_t = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_t\right)$

The loss term $\mathcal{L}_{t-1}$ is revisited to minimize the difference between μ_θ and $\tilde{\mu}$

$$\mathcal{L}_{t-1} = \mathbb{E}_{x_0 \sim q, \epsilon \sim \mathcal{N}} \left[\frac{(1-\alpha_t)^2}{2\alpha_t(1-\bar{\alpha}_t)\bar{\beta}_t^{\,2}} \left\| \epsilon_t - \epsilon_\theta(x_t, t) \right\|^2 \right]$$

➔ This expression can be simplified by ignoring the weighting term, which gives the final loss function to minimize as follows :

$$\mathcal{L}_{t-1}^{simple} = \mathbb{E}_{x_0 \sim q, \epsilon \sim \mathcal{N}, t \sim [1,T]} \left[\left\| \epsilon_t - \epsilon_\theta(x_t, t) \right\|^2 \right]$$

DDPM

Architecture

- Key points

  - The goal is to estimate the conditional probability $p_\theta(x_{t-1}|x_t)$

$$p_\theta(x_{t-1} \mid x_t) = \mathcal{N}(\mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$$

$$\begin{cases} \mu_\theta(x_t, t) = \dfrac{1}{\sqrt{\alpha_t}}\left(x_t - \dfrac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t)\right) \\ \Sigma_\theta(x_t, t) = \sigma_t \mathbf{I} = \tilde{\beta}_t \mathbf{I} \end{cases}$$

  - Although the key modeling of diffusion models is the Markov chain, it is possible to directly express x_t as a function of x_0

$$x_t = \sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1-\bar{\alpha}_t}\, \epsilon_t$$

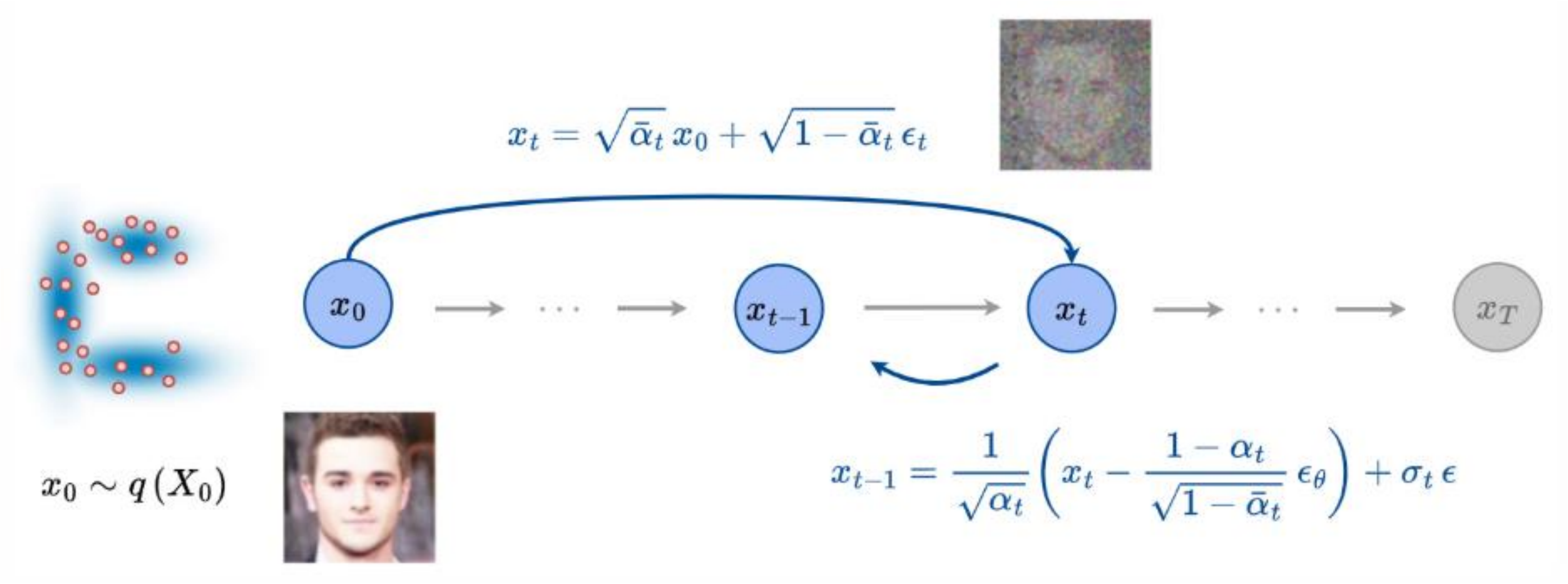
$$\begin{cases} \alpha_t = 1 - \beta_t \\ \bar{\alpha}_t = \prod_{k=1}^{t} \alpha_k \end{cases} \quad \text{and} \quad \epsilon_t = \mathcal{N}(0, \mathbf{I})$$

  - The only unknown is the noise $\epsilon_\theta(x_t, t)$, which we will estimate using a neural network by minimizing the following loss function

$$\mathcal{L}_{t-1}^{simple} = \mathbb{E}_{x_0 \sim q, \epsilon \sim \mathcal{N}, t \sim [1,T]}\left[\left\|\epsilon_t - \epsilon_\theta(x_t, t)\right\|^2\right]$$

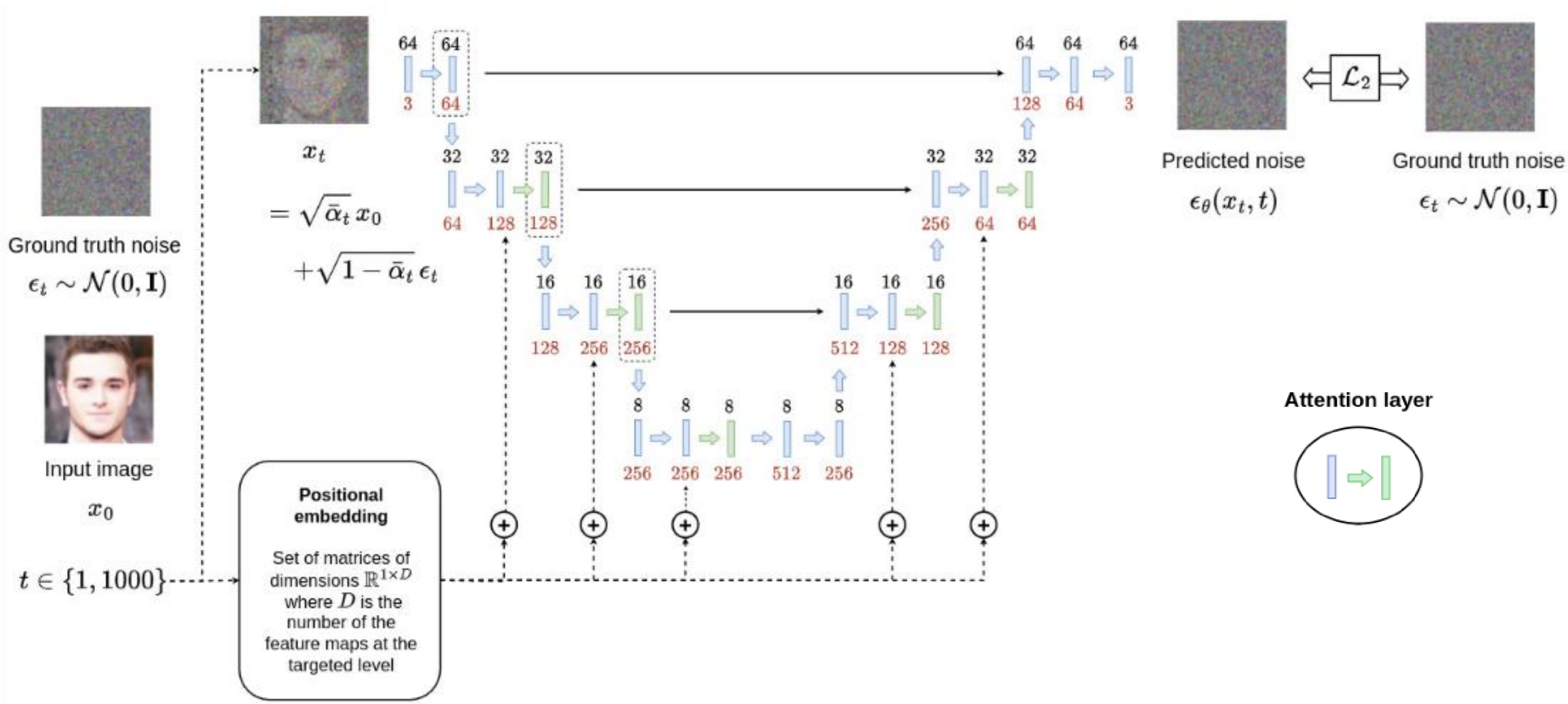
It is therefore possible, at any time step t, to generate a noisy image x_t from x_0 and ϵ_t, which are known, and learn to estimate ϵ_t from x_t

The estimated noise $\epsilon_\theta(x_t, t)$ can then be used to recover x_{t-1} from x_t



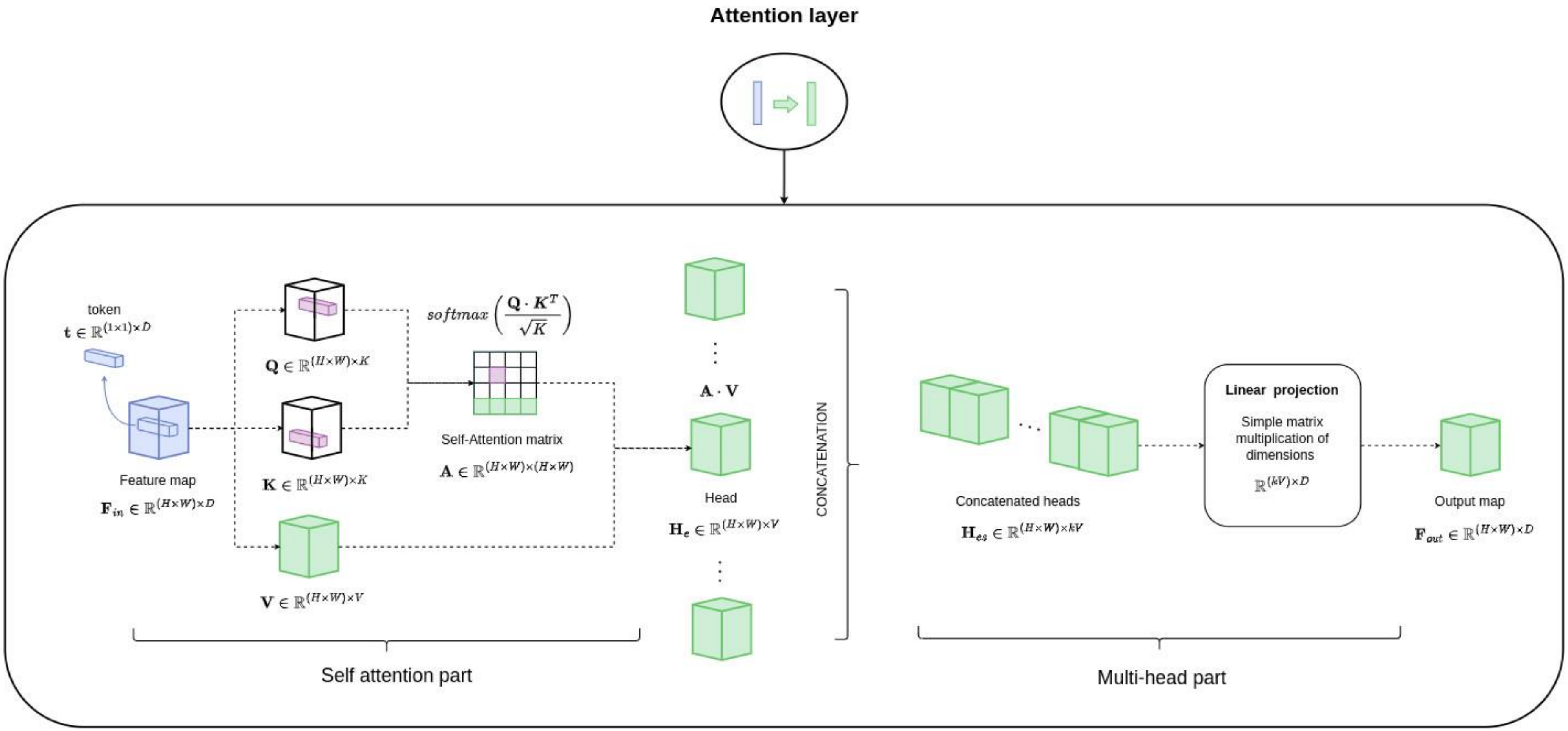
Standard U-Net with attention layers and position encoding to integrate temporal information

→ Integration of t is necessary because the added noise varies over time



Architecture

➔ Attention layer



- In summary
  - ➔ Training

Algorithm 1 Training

repeat
  $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
  $t \sim \mathrm{Uniform}(\{1, \ldots, T\})$
  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  Take gradient descent step on
  $$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
until converged

  - ➔ Inference / generation of a new synthetic image

Algorithm 2 Sampling

$\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
for $t = T, \ldots, 1$ **do**
  $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
  $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right) + \sigma_t \mathbf{z}$
end for
return $\mathbf{x}_0$

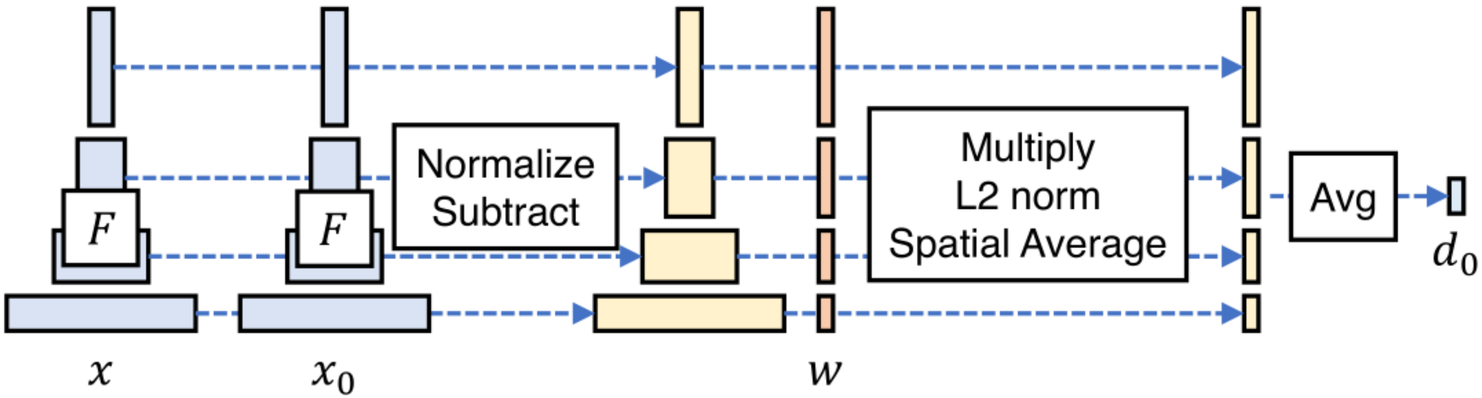
Practical application

Latent diffusion models

- Projection of images into a dedicated space before processing

  → Using a VAE as input/output to the DDPM to reduce the complexity of the processed images and memory footprint

  → Introducing a perceptual loss function to improve the quality of the reconstructed images

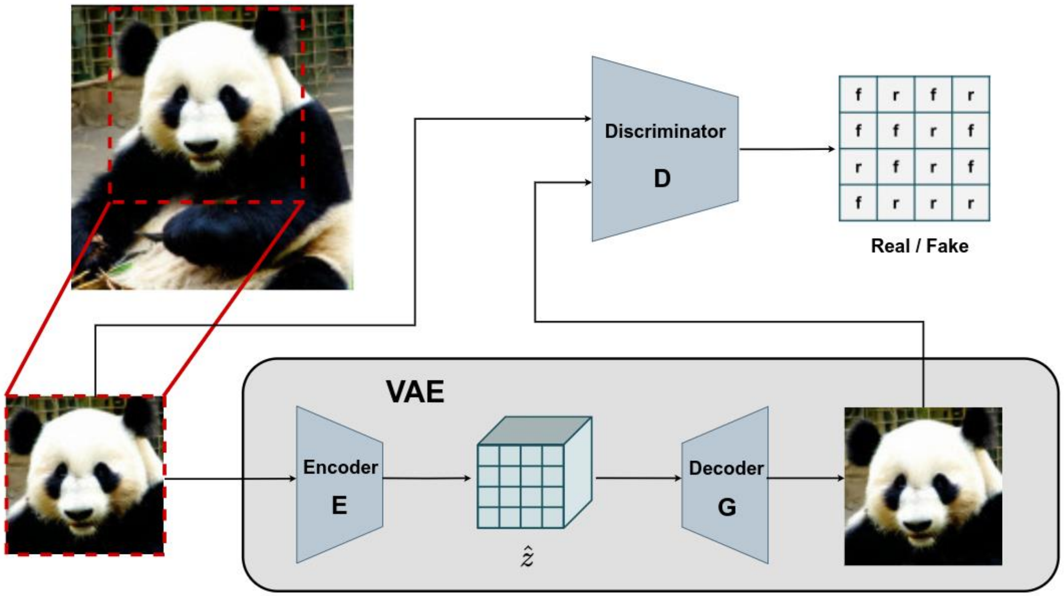


x and x_0 are two image patches given as input

F is a pre-trainer network, such as VGG50

- Image projection in a dedicated area before processing

→ Implementation of an adversarial approach



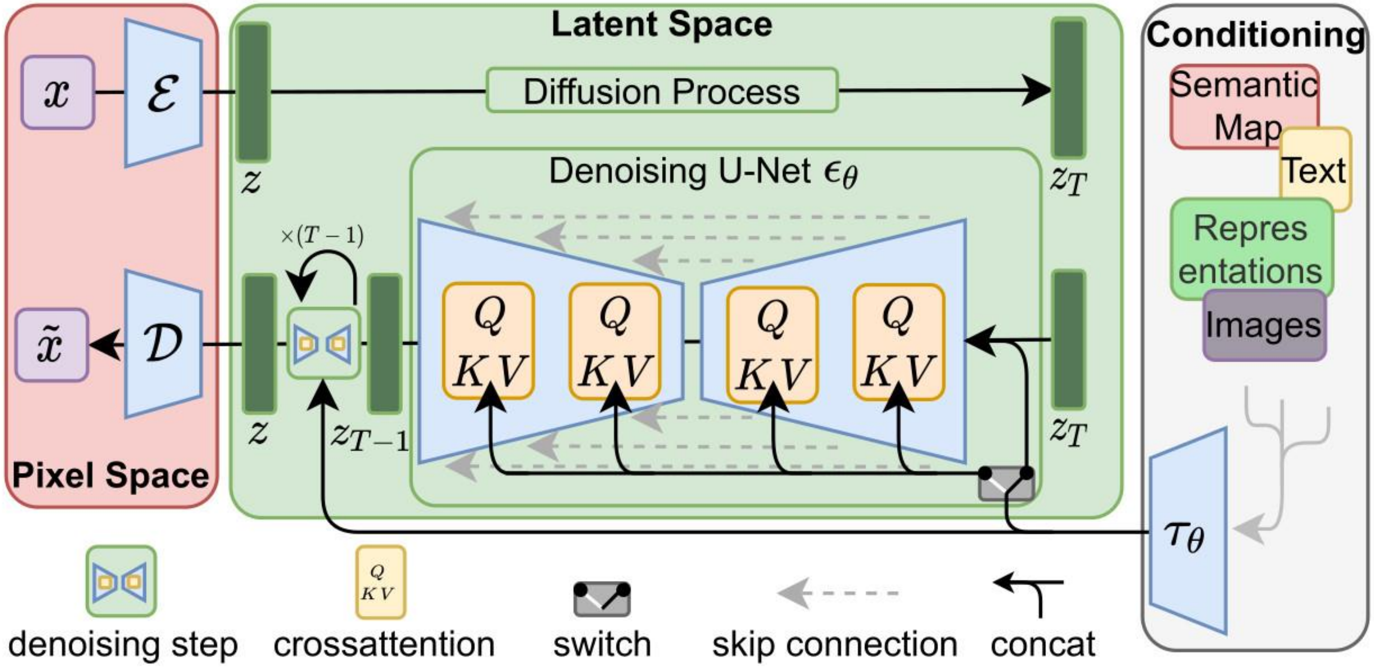
→ Final loss function

$$\mathcal{L} = \mathcal{L}_{recons} + \beta_1 \mathcal{L}_{KLD} + \beta_2 \mathcal{L}_{perceptual} + \beta_3 \mathcal{L}_{adversarial}$$

- VAE is learned independently of DDPM and its architecture is fixed
- Minimization of the following loss function

$$\mathcal{L}_{LDM} = \mathbb{E}_{x_0 \sim q, \epsilon \sim \mathcal{N}, t \sim [1,T]} \left[\|\epsilon_t - \epsilon_\theta(x_t, t)\|^2 \right]$$

- LDM architecture



- Properties

Parameters	LDM – 256×256
z-shape	$64 \times 64 \times 3$
Diffusion steps	1000
Noise scheduler	linear
Number of parameters	274 Million
Channels	224
Channel multiplier	$1, 2, 3, 4$
Attention resolutions	$32, 16, 8$
Number of head	1
Batch size	48
Iterations	410 k
Learning rate	$9.6\, e^{-5}$

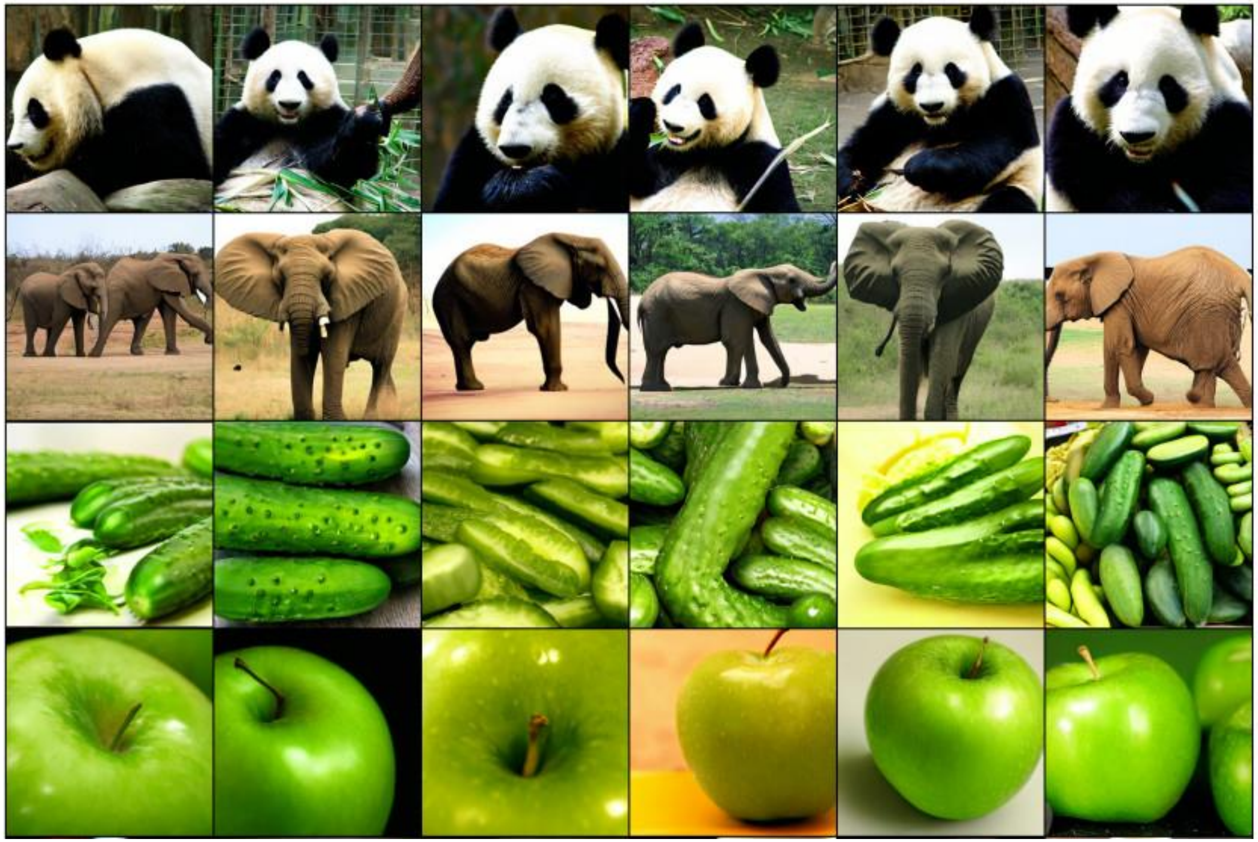
- Random generation of synthetic images *without conditioning* learned from the CelebA-HQ database

Random samples on the CelebA-HQ dataset

Latent diffusion model (LDM)

- Random generation of synthetic images *with conditioning on the class* learned from the ImageNet database

Random class conditional samples on the ImageNet dataset



- Random generation of synthetic images *with conditioning on masks* learned from the Flickr-landscapes database

Latent diffusion model (LDM)

- Random generation of synthetic images *with conditioning on text* learned from LAION-400M database

  - ➔ Using the BERT tokenizer
  - ➔ This model has over 1.45 billion parameters!

'A painting of the last supper by Picasso.'

That's all folks